# Teaching Security Requirements Engineering Using SQUARE

Nancy R. Mead, Software Engineering Institute [vita[1]]
Dan Shoemaker, University of Detroit Mercy [vita[2]]
Jeffrey A. Ingalsbe, Ford Motor Co. [vita[3]]

2011-02-15

This paper details the validation of a comprehensive teaching model for security requirements engineering which ensures that security is built into the software from its inception. It centers on the employment of the SQUARE method for secure software requirements engineering, which was developed at Carnegie Mellon University. The effectiveness of the SQUARE method, its learning system and the initial results of using it in student case studies and in a practical, higher education classroom application are reported.

## 1. Introduction

The software that underpins critical infrastructure has to be secure. Yet, the problem is that historically it has been almost impossible to build secure software. As a result, it is estimated that the exploitation of unsecured software costs the U.S. economy an average of $60 billion dollars per year [1].

Notwithstanding the importance of financial loss however, the real concern lies in the fact that the exploitation of a flaw in the software that underlies basic infrastructure services like power and communication could cause a significant disaster [2]. The U.S. Critical Infrastructure Taskforce sums up that likelihood in a single statement: “A digital disaster strikes some enterprise every day, [and] infrastructure disruptions have cascading impacts, multiplying their cyber and physical effects” [3, p. 6].

Because of the key importance of practitioners trained in secure software, The National Strategy to Secure Cyberspace – Action/ Recommendation 2-14 has mandated the Department of Homeland Security (DHS) to “promulgate best practices and methodologies that promote integrity, security, and reliability in software code development, including processes and procedures that diminish the possibilities of erroneous code, malicious code, or trap doors that could be introduced during development” [4, p. 35].

The global solution is to ensure that secure practice is being followed in all aspects of traditional lifecycle work. However in order to realize this goal, it is essential to educate the professional community in the particular concepts and methods of secure practice. Thus, it is important to develop targeted content and focused instruction that will both ensure the proper understanding of secure software engineering practice, as well as reinforce its importance to software engineering students.

It would seem to be a simple task to “identify the necessary workforce competencies, leverage sound practices, and guide curriculum development for education and training relevant to software assurance” [5, p. xiv]. However, the problem is that security is not a mature field, and so the teaching of security topics is done in a number of disjointed places within higher education. That includes “software engineering, systems engineering, information systems security engineering, safety, security, testing, information assurance, and project management” [5, p. xiv].

Coherent knowledge about “software assurance processes and practices has yet to be integrated into the body of knowledge of the contributing disciplines” [5, p. xiv]. Too often, the result of this lack of integration is the graduation of a software engineering student who develops buggy code with weak security measures.

It is both impractical and impossible to simply drop the whole body of software assurance knowledge into a traditional computer curriculum.

1. http://buildsecurityin.us-cert.gov/bsi/about_us/authors/230-BSI.html (Mead, Nancy)
2. http://buildsecurityin.us-cert.gov/bsi/about_us/authors/689-BSI.html (Shoemaker, Dan)
3. http://buildsecurityin.us-cert.gov/bsi/about_us/authors/687-BSI.html (Ingalsbe, Jeff)

Therefore it seems important to adopt a focused strategy and a clear starting point. One of the logical places to begin the integration process is in an area that is vital to good security practice, but which is also well established and important to general development. That is security requirements engineering.

## 2. The importance of requirements engineering

It is well recognized that requirements engineering is critical to the success of any major development project [6, 7, 8, 9, 10]. Several authoritative studies have shown that requirements engineering defects cost 10 to 200 times as much to correct once fielded than if they were detected during requirements development [11]. Other studies have shown that reworking requirements defects on most software development projects costs 40 to 50 percent of total project effort, and the percentage of defects originating during requirements engineering is estimated at more than 50 percent [12, 13]. The total percentage of project budget due to requirements defects is 25 to 40 percent [12, 13].

Microsoft has indicated that versions of Windows developed after the Microsoft Security "Push" have half the patch levels of earlier versions of Windows, an obvious savings. Other recent industry data suggests that vulnerabilities cost up to 100 times less to correct when they are found during security requirements engineering, rather than after a system is operational. Thus the costs of poor security requirements show that even a small improvement in this area would provide a high value. By the time that an application is fielded and in its operational environment, it is very difficult and expensive to significantly improve its security.

According to an overwhelming number of studies [6, 7, 8, 11, 12, 14, 15, to name a few], requirements problems are the number one cause of why projects

- are significantly over budget
- are significantly past schedule
- have significantly reduced scope
- deliver poor-quality applications
- are not significantly used once delivered
- are cancelled

## 3. The problem with developing security requirements

Security requirements are often identified during the system life cycle. However, the requirements tend to be general mechanisms selected from standard lists, such as password protection, firewalls, and virus detection tools. Often the security requirements are developed independently of the rest of the requirements engineering activity and hence are not integrated into the mainstream of the requirements activities. As a result, security requirements that are specific to the system and that provide for protection of essential services and assets are often neglected.

In reviewing requirements documents, we typically find that security requirements, when they exist, are in a section by themselves and have been copied from a generic set of security requirements. The requirements elicitation and analysis that is needed to get a better set of security requirements seldom takes place.

Much of the study of requirements engineering research and practice has addressed the capabilities that the system will provide. So a lot of attention is given to the functionality of the system, from the user's perspective, but little attention is given to what the system should *not* do. For instance, in one discussion on requirements prioritization for a specific large system, ease of use was assigned a higher priority than security requirements.

A key problem is that, if security requirements are not effectively defined, the resulting system cannot be effectively evaluated for success or failure prior to implementation. Security requirements are often missing in the requirements elicitation process and tend to be neglected subsequently. In addition to employing applicable software engineering techniques, the organization must understand how to incorporate the techniques into its existing software development processes [16]. That is the reason to embed security requirements instruction into general courses in requirements elicitation and documentation. The question is, “how best to do that”.

## 4. Integrating security requirements into standard curricula

A number of approaches can be used for integrating security requirements into standard curricula. At the National Institute of Informatics in Japan, the Top SE program [17] includes security requirements engineering as part of its curriculum. The Top SE program includes discussion of misuse cases, TROPOS [18], and goal-driven requirements engineering (KAOS) [19]. In addition there is a case study based on the Common Criteria.

Case studies for security requirements engineering and security engineering in general have been used at the International Institute of Information Technology, Hyderabad [20] as a means of bridging the industry/ university gap.

The CERT program at the Software Engineering Institute has, over three academic semesters, experimented with a novel technique to educate students on the development of security requirements engineering for software systems [9].

This paper reports the findings of a couple of studies designed to validate a comprehensive teaching model for requirements definition, which ensures that security is built into the software from its inception. It centers on the employment of the SQUARE method of secure software requirements definition, which was developed at Carnegie-Mellon University. The effectiveness of the SQUARE method, its learning system and the initial results of using it in a practical, higher education classroom application will be reported.

## 5. Case study one: Integrating security requirements into SWE curricula

The motivation behind SQUARE is to see whether good requirements engineering processes can be adapted specifically to the problem of identifying security requirements. If this can be done successfully, organizations will have the ability to identify security requirements up front rather than as an afterthought. The SQUARE process provides a means for eliciting, categorizing, and prioritizing security requirements for information technology systems and applications. Note that while there is nothing unique about the steps in the process, which have existed for many years in requirements engineering, we have seen relatively little evidence of their application to security requirements, and even less on whether such a process is successful for developing security requirements.

The existing methods fit nicely into the SQUARE process. These include misuse and abuse cases, attack trees, and formal methods. Others, such as the Common Criteria and SCR, suggest their own requirements engineering process. The SQUARE methodology seeks to build security concepts into the early stages of the development life cycle. The model may also be useful for documenting and analyzing the security aspects of fielded systems and could be used to steer future improvements and modifications to these systems. The process is best applied by the project’s requirements engineers and security experts in the context of supportive executive management and stakeholders. We believe the process works best when elicitation occurs after risk assessment (Step 4) has been done and when security requirements are specified prior to critical architecture and design decisions. Thus, critical business risks will be considered in the development of the security requirements. The SQUARE steps are summarized below. A detailed discussion of SQUARE and how to apply it can be found in [21].

1. **Agree on Definitions**, is needed as a prerequisite to security requirements engineering. On a given project, team members will tend to have definitions in mind, based on their prior experience, but those definitions will not necessarily agree [22]. For example, to some government organizations, security has to do with access based on security clearance levels, whereas to others security may have to do with physical security or cyber security. It is not necessary to invent definitions. Most likely, sources such as IEEE and SWEBOK will provide a range of definitions to select from or tailor. A focus group meeting with the interested parties will most likely allow a consistent set of definitions to be selected for the security requirements activity.

2. **Identify Assets and Security Goals** – this step should be done at the level of the organization and is needed to develop the information system. This provides a consistency check with the organization's policies and operational security environment. Different stakeholders will likely have different ideas about which assets are important and the associated security goals. For example, a stakeholder in human resources may be concerned about maintaining the confidentiality of personnel records, whereas a stakeholder in a financial area may be concerned with ensuring that financial data is not accessed or modified without authorization. It is important to have a representative set of stakeholders, including those with operational expertise. Once the assets and goals of the various stakeholders have been identified, the goals will need to be prioritized. In the absence of consensus, an executive decision may be needed to prioritize these goals.
3. **Develop Artifacts**, is necessary to support all the subsequent activities. It is often the case that organizations do not have a documented concept of operations for a project, succinctly stated project goals, documented normal usage and threat scenarios, misuse cases, and other documents needed to support requirements definition. This means that either the entire requirements process is built on a foundation of sand, or a lot of time is spent backtracking to try to obtain such documentation.
4. **Perform Risk Assessment**, requires an expert in risk assessment methods, the support of the stakeholders, and the support of a requirements engineer. There are a number of risk assessment methods available. A specific method can be recommended by the risk assessment expert, based on the needs of the organization. The artifacts from Step 3 provide the input to the risk assessment process. The outcomes of the risk assessment can help in identifying the high-priority security exposures. Organizations that do not perform risk assessment typically do not have a logical approach to consider organizational risk when identifying security requirements but tend to select mechanisms, such as encryption, without really understanding the problem that is being solved.
5. **Select Elicitation Technique**, becomes important when there are several classes of stakeholders. A more formal elicitation technique, such as JAD or structured interviews, can be effective in overcoming communication issues when there are stakeholders with different cultural backgrounds. In other cases, elicitation may simply consist of sitting down with a primary stakeholder to try to understand that stakeholder's security requirements needs.
6. **Elicit Security Requirements**, is the actual elicitation process using the selected technique. Most elicitation techniques provide detailed guidance on how to perform elicitation. This builds on the artifacts that were developed in earlier steps, such as misuse and abuse cases, attack trees, threat scenarios, etc.
7. **Categorize Requirements**, allows the requirements engineer to distinguish among essential requirements, goals (desired requirements), and architectural constraints that may be present. Requirements that are actually constraints typically occur when specific system architecture has been chosen prior to the requirements process. This is good, as it allows assessment of the risks associated with these constraints. This categorization also helps in the prioritization activity that follows.
8. **Prioritize Requirements**, depends not only on the prior step, but may also suggest performing a cost/ benefit analysis in order to determine which security requirements have a high payoff relative to their cost.
9. **Inspect Requirements**, can be done at varying levels of formality, from Fagan Inspections to peer reviews. Once inspection is complete, the organization should have an initial set of prioritized security requirements. It should also understand which areas are incomplete and must be revisited later. Finally, the organization should understand which areas are dependent on specific architectures and implementations, and expect to revisit those as well.

In student team projects over three separate semesters, thirteen students gained hands-on experience through case applications involving real-world software development projects. Using SQUARE [21], the students were able to understand the importance of security requirements in software systems, as well as improve the security foundations of the client projects with which they worked.

The students had a variety of backgrounds in our academic case study group. Some had background in security and some had background in software engineering or information technology. However, none of the students had experience in eliciting and documenting security requirements for software systems. They also did not have experience working with software engineering methods, such as SQUARE. The students had to develop two products to complete their course requirements: (1) a document that was delivered to the client proposing security requirements and supporting artifacts for the client's project and (2) a process document delivered only to the faculty advisor. This second document discussed how the students went about applying each step in the process, whether it was easy or difficult to apply, and how it could be improved on. To that end, the project provided them with a unique learning opportunity. The following student feedback shows how the SQUARE experience helped them later on in the workforce:

Student 1: "The real-world experience I gained from the SQUARE project gave me the perfect set of information security project management and budgeting skills that were invaluable in my job."

Student 2: "While working on the SQUARE project with Dr. Mead, I took part in several in-depth case studies involving organizations of varying size and reputation. It was a wonderful opportunity to get a feel for how real companies develop and manage large IT projects. This insight, along with the security focus of SQUARE, allowed me to hit the ground running here with the security projects we're developing. Overall it was an extremely valuable experience and I'm grateful that I was involved."

## 6. Case study two: Assessment of increase in security knowledge

The SQUARE methodology was presented to students in a basic, graduate software engineering requirements modeling class. The class format is basically stand-up lectures with student projects. The study involved two groups of students. The participants were all graduate students in a software engineering management program at University of Detroit Mercy, as described in [23]. These were all advanced students. Most of them had had the software management core, with the exception of the specification class. That core is comprised of project management, software processes, object modeling, and a testing and assurance class.

The students all had an acceptable amount of academic background in undergraduate software engineering, information systems, or professional IT work. The students in the sample groups were formed from classes that were part of the regularly scheduled curriculum. The groups sampled comprised both fall and winter offerings of these two. Because class sizes varied, the actual number of students in the groups also varied. However, the number was uniformly comparable within each course. Thus, the individual group sizes were between 12 and 15 for the treatment.

One group was given the security requirements engineering and SQUARE preparation. This consisted of four lectures as follows: A general introduction to security requirements engineering, An overview of the SQUARE method and steps, Detailed discussion of SQUARE steps 1-4, and Detailed discussion of SQUARE steps 5-9. These lectures are available for download from the CERT website (http://www.cert.org/sse/square.html). The control group was given the normal set of lectures, which are based on the contents of IEEE Standard 830-1993 IEEE Recommended Practice for Software Requirements Specifications [24].

Each group was given a pretest of knowledge involving 9 multiple choice questions (in Appendix A). Then, following the administration of the SQUARE preparation the same two groups were given a post-test containing the same questions and their products. The results were then subjected to a Student t http://en.wikipedia.org/wiki/Student's_t-distribution[5] in order to determine whether the SQUARE lectures led to an increased capability in security requirements definition.

Table 1 displays the results of that analysis for both groups. In Table 1, df is degrees of freedom, p-value is probability of significance, t is student t value:

| Question # | Pre-Test | Post-Test | Pre-Test | Post-Test |
|---|---|---|---|---|
| 1 | 0.18 | 0.19 | 0.28 | 0.59 |
| 2 | 0.75 | 0.73 | 0.84 | 0.73 |
| 3 | 0.58 | 0.30 | 0.60 | 0.73 |

5. http://en.wikipedia.org/wiki/Student%27s_t-distribution

| | | | | |
|---|---|---|---|---|
| 4 | 0.67 | 0.66 | 0.44 | 0.73 |
| 5 | 0.67 | 0.71 | 0.44 | 0.55 |
| 6 | 0.42 | 0.52 | 0.44 | 0.50 |
| 7 | 0.50 | 0.52 | 0.68 | 0.68 |
| 8 | 0.58 | 0.65 | 0.64 | 0.68 |
| 9 | 0.75 | 0.77 | 0.56 | 0.59 |
| | | | | |
| Mean | 0.57 | 0.56 | 0.55 | 0.64 |
| StDev | 0.18 | 0.20 | 0.17 | 0.09 |
| | | | | |
| df=12 | Student t= | 0.11 | Student t= | -1.40 |
| | p-value= | 0.54 | p-value= | 0.09 |

**Table 1: Analysis of Pre and Post Test Results for Control and Treatment Groups**

Neither group achieved significance. However, the large difference in p value between the control and the treatment group would indicate that the SQUARE treatment provided a considerable increase in security knowledge. One explanation for the fact that the treatment did not quite achieve significance was the fact that both groups were composed of advanced software engineering students and as a result it is likely that both groups had inherent awareness of security topics.

As an additional test of the effectiveness of the SQUARE treatment, the student’s end of term project deliverables were subjected to a qualitative analysis by the instructor. The instructor has taught this same course for the past seven years. It was found that the quality of the student’s project deliverables increased greatly. The use-case model and the analysis model for the semester model addressed security requirements and risk in a way that was far superior to other semesters. Additionally, the students were able to achieve these results with less intervention and help from the instructor.

## 7. Practical implications and future plans

It’s easy to see that this idea could be extended to include software engineering practices that focus on developing secure software. Although these are early results, based on two cases, when students learn about security requirements engineering and SQUARE, they have a better understanding of what is needed to produce more secure software. That finding is potentially important to the field of software engineering. As such, the validated SQUARE model could prove both highly useful to the profession as well as potentially very influential in the teaching of security requirements engineering practice. It’s easy to see that this idea could be extended to include software engineering practices that focus on developing secure software.

As experience with these approaches grows, our plans include the gathering of more quantitative data to show the benefit of the approach we have discussed here. So far there have been some 300 downloads of the SQUARE educational material from the CERT website. One of our goals this year is to conduct a survey to find out about the usage of the material and its results. It is our hope that in the future there will be more synergy between software assurance and software engineering education. Other universities around the world offer courses and lectures that include methods for developing secure software. If those universities conduct similar studies, we expect to see additional results. For example, SQUARE educational material was translated into Chinese and delivered at National Defence University in Taiwan. Feedback from this offering is forthcoming. If more universities incorporate the content of the 4 SQUARE lectures into their course offerings, we hope to strengthen the results presented here.

## References

1. M. Newman, “Software errors cost U.S. economy $59.5 billion annually,” National Inst. of Standards and Technology (NIST), Gaithersburg, MD, 2002.
2. R. A. Clark, *Breakpoint*. New York, NY: G. P. Putnam and Sons, 2007.
3. R. A. Clark and H. A. Schmidt, “A national strategy to secure cyberspace,” The President’s Critical Infrastructure Protection Board, Washington, DC, 2002.
4. National Infrastructure Advisory Council (NIAC). “National strategy to secure cyberspace.” Washington, DC: U.S. Department of Homeland Security, 2003.
5. S. T. Redwine (Ed.), “Software assurance: A guide to the common body of knowledge to produce, acquire and sustain secure software, version 1.1.” U.S. Department of Homeland Security, Washington, DC, 2006.
6. T. Addison and S. Vallabh, “Controlling software project risks – an empirical study of methods used by experienced project managers,” KPMG, 2000.
7. J. J. Carr, “Requirements engineering and management: The key to designing quality complex systems,” *The TQM Magazine,* vol. 12, pp. 400-407, November-December 2000.
8. H. Hecht and M. Hecht, “How reliable Are requirements for reliable software?” *Software Tech News,* vol. 3, 2000.
9. N. R. Mead and E. Hough, “Security requirements engineering for software systems: Case studies in support of software engineering education,” in *19th Conf. Software Engineering Education and Training,* Turtle Bay, Hawaii, 2006, pp. 149-156.
10. B. Palyagar, “Measuring and influencing requirements engineering process quality,” in Australian Workshop on Requirements Engineering, Adelaide, Australia, 2004.
11. B. Boehm and V. Basili, “Software defect reduction – Top 10 list,” *IEEE Computer,* vol. 34, pp. 135-137, January 2001.
12. T. McGibbon, “A business case for software process improvement revised.” Washington, DC: DoD Data Analysis Center for Software (DACS), 1999.
13. N. R. Mead and T. Stehney, “Security Quality Requirements Engineering (SQUARE) methodology,” in *SESS '05: 2005 Workshop on Software Engineering for Secure Systems—Building*

*Trustworthy Applications*, St. Louis, MO, 2005. (2005b)

14. S. Lauesen and O. Vinter, "Preventing requirement defects: An experiment in process improvement," *Requirements Engineering Journal*, vol. 6, February 2001, pp. 37-50.

15. B. Palyagar, "A framework for validating process improvements in requirements engineering, in *RE 04-IEEE Joint Int.*, 2004, pp. 33-36.

16. R. C. Linger, N. R. Mead, and H. F. Lipson, "Requirements definition for survivable systems," in Third International Conf. Requirements Engineering, 1998, pp. 14-23.

17. S. Honiden, Y. Tahara, N. Yoshioka, K. Taguchi, and H. Washizaki, "Top SE: Educating superarchitects who can apply software engineering tools to practical development in Japan," in *29th International Conf. Software Engineering (ICSE'07)*, 2007, pp. 708-718.

18. P. Giorgini, H. Mouratidis, and N. Zannone, "Modeling security and trust with Secure Tropos," in *Integrating Security and Software Engineering: Advances and Future Visions*. Hershey, PA: IGI Global, 2007, pp. 160-189.

19. R. de Landtsheer and A. van Lamsweerde, "Reasoning about confidentiality at requirements engineering time," in *10th European Software Engineering Conf.*, Lisbon, Portugal, 2005, pp. 41-49.

20. K. Garg and V. Varma, "Security: Bridging the academia-industry gap using a case study," in *XIII Asia Pacific Software Engineering Conf.*, Bangalore, India, 2006.

21. N. R. Mead, E. Hough, and T. Stehney, "Security Quality Requirements Engineering (SQUARE) methodology," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU/SEI-2005-TR-009, 2005.

22. C. Woody, "Eliciting and Analyzing Quality Requirements: Management Influences on Software Quality Requirements," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU/SEI-2005-TN-010, 2005.

23. G. Ford, "A progress report on undergraduate software engineering education," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU/SEI-94-TR-011, 1994.

24. IEEE Computer Society, “IEEE recommended practice for software requirements specifications,” IEEE Standard 830-1993, Oct. 1998.

## Carnegie Mellon University and IEEE Copyright

1. mailto:permission@sei.cmu.edu